PhD Candidacy Exam:
Host and Network Defense Systems For Intrusion Reaction

Michael E. Locasto
{*locasto@cs.columbia.edu*}
522 CSB
Department of Computer Science
Columbia University
New York, NY 10027

November 7, 2004

Abstract

The focus of this candidacy exam is the investigation of the state of the art in intrusion reaction systems. *Intrusion reaction*, the design and careful selection of mechanisms to automatically respond to network attacks, has recently received an amount of attention that rivals its equally difficult sibling intrusion detection. Response systems vary from the low–tech (manually shut down misbehaving machines) to the highly ambitious (on the fly "vaccination", validation, and replacement of infected software). In the middle lies a wide variety of practical techniques, promising technology, and nascent research.

1 Purpose

"The candidacy exam certifies that the student has demonstrated a depth of scholarship in the literature and the methods of the student's chosen area of research, and has demonstrated a facility with the scholarly skills of critical evaluation and verbal expression." [1]

The tentative date for the exam is 30 November 2004.

2 Candidate Research Area Statement

Intrusion reaction is the careful, rational, and automatic selection of an appropriate response to the threat or event of system penetration or subversion. This ability is predicated on the availability of a trusted computing base in hosts and networks – an uncorrupted environment in which recovery actions can execute.

Intrusion reaction (IR) has been given less attention than it deserves due to several critical issues. First, the limits of detection technology have historically mandated that the shortcomings of intrusion detection (false positives, fail–open nature, performance) be addressed before a reaction can take place – an attack must be detected before any response can be mounted. Second, many system administrators and policy coordinators are understandably hesitant about transferring control of the network and host systems to a machine, even though (and perhaps because) a machine can react orders of magnitude faster than a human system administrator. Third, most reaction systems are insufficiently advanced enough to respond intelligently to attacks.

For example, a simple response system is a packet–filter firewall that will dynamically update its rule base to close off certain ports in response to attack traffic recognized by an IDS[2]. A system with this mechanism at its core is proposed in [41]. Even this straightforward model has its limitations: simple source address spoofing can result in denying

[1] `http://lenox.psl.cs.columbia.edu/phdczar/candidacy.html`
[2] `http://www.mcafeesecurity.com/us/_tier2/products/_media/mcafee/ds_intrushieldidssensor.pdf`

service to legitimate sites, and other TCP tricks can bypass the response mechanism[3]. Another protection mechanism is based on intercepting patterns of system calls. McAfee is marketing a product that employs this technology[4].

3 Faculty Candidacy Committee Members

The candidate respectfully solicits the guidance and expertise of the following faculty members and welcomes suggestions for other important papers and publications in the exam research area.

1. Angelos Keromytis
2. Sal Stolfo
3. JI

4 Exam Scope and Structure

The scope of this exam is narrowly focused on state–of–the–art mechanisms that provide a basis for flexible and correct reaction to attacks or system subversion. The unifying theme of the systems in this study is that they contain or describe a mechanism that can automatically nullify or mitigate the effects of an attack or exploit during runtime. Most of these systems can accomplish this response without previous knowledge of an exploit or particular vulnerability. We acknowledge that these systems have as their primary goal service and system survivability. There is a rich literature on redundancy and diversity to support system survivability, but the focus of this collection is on systems that recognize attacks and adapt at runtime.

Furthermore, the focus of this exam is not on methods of validation or verification of responses and reconfigured systems, nor does it examine the extensive literature on replication and fault tolerance. The closest "verification" papers are Demsky's data structure repair [14], Naldurg's dynamic access control policy [28], and Kreidl's feedback control [26].

The following papers are listed in groupings by title with both an abstract and a reference to the full citation in the attached bibliography.

4.1 Background Material

1. **How Re(Pro)active Should an IDS Be?** [31]
   Abstract: *The classical security paradigm of Protect, Detect, React has traditionally been applied to the field of information security with Firewalls taking on the role of protection while detection is handled by Intrusion Detection Systems (IDS). This admittedly simplistic picture leaves open two questions: who or what should react? and how?*

2. **Intrusion Reaction: Recommendations for Obtaining Reaction Capabilities** [27]
   Abstract: *The Command and Control (C2) Protect Mission-Oriented Investigation & Experimentation (MOIE) Project, sponsored by the Air Force, develops and promulgates resources to counter information warfare (IW) threats to military C2 computer networks. This report has been produced by the Intrusion Reaction task of the project. A growing threat to Air Force networks and computers is exploitative intrusion activity. One technological countermeasure to exploitative intrusion activity is intrusion reaction capability. But intrusion detection and reaction (IDR) systems in operation today do not provide a number of reaction features that might materially help the Air Force protect its networks and computers. This report develops a profile of such features. It recommends areas where the Air Force can make effective investments in research, development, and investigation of intrusion reaction capabilities that can improve IDR systems.*

3. **Intrusion Detection and Isolation Protocol: Automated Response to Attacks** [41]
   Abstract: *With current intrusion detection technology, it is possible to detect attacks in real time. Typically,*

[3] `http://www.securityfocus.com/infocus/1540`
[4] `http://www.mcafeesecurity.com/us/products/mcafee/host_ips/standard_edition.htm`

when an IDS system is triggered, a human operator is notified. The system is then adjusted manually in response to the intrusion. Many types of attacks, however, can only be thwarted if the response is quick. Responding quickly, without considering the self-inflicted damage that a response might inadvertently cause, however, is equally dangerous. Should the attacker discover the response mechanism, a willful triggering could be a denial-of-service attack in itself. Our Intrusion Detection and Isolation Protocol (IDIP) automates attack response. IDIP is a protocol whereby filtering routers, firewalls and host-based response modules cooperate with intrusion detection systems (IDS) to trace the attack back to the source and stop it. To be able to initiate a quick response that halts an attack in progress, while still maintaining the ability to issue the optimum response minimizing self-inflicted damage, our system employs a staged response approach.

4. **The Proactive Security Toolkit and Applications** [5]
   Abstract: *Existing security mechanisms focus on prevention of penetrations, detection of a penetration, and (manual) recovery tools. Indeed, attackers focus their penetration efforts on breaking into critical modules, and on avoiding detection of the attack. As a result, security tools and procedures may cause the attackers to lose control over a specific module (computer, account), since the attacker would rather lose control than risk detection of an attack. While controlling the module, the attacker may learn critical secret information or modify the module that make it much easier for the attacker to regain control over that module later. Recent results in cryptography give some hope of improving this situation; they show that many fundamental security tasks can be achieved with* proactive security. *Proactive security does not assume that there is any module completely secure against penetration. Instead, we assume that at any given time period (day, week, ...) a sufficient number of the modules in teh system are secure (not penetrated). The results obtained so far include some of the most important cryptographic primitives such as signatures, secret sharing, and secure communication. However, there was no usable implementation, and several critical issues (for actual use) were not addressed.*
   In this work we report on a practical toolkit implementing the key proactive security mechanisms. The toolkit provides secure interfaces to make it easy for applications to recover from penetrations. The toolkit also addresses other critical implementation issues, such as the initialization of the proactive secure system. We describe the toolkit and discuss some of the potential applications. Some applications require minimal enhancements to the existing implementations - e.g. for secure logging (especially for intrusion detection), secure end-to-end communication and timestamping.

5. **A Holistic Approach to Service Survivability** [23]
   Abstract: *We present SABER, a proposed survivability architecture that blocks, evades, and reacts to a variety of attacks by using several security and survivability mechanisms in an automated and coordinated fashion. Contrary to the ad hoc manner in which contemporary survivable systems are built-using isolated, independent security mechanisms such as firewalls, intrusion detection systems and software sandboxes – SABER integrates several different technologies in an attempt to provide a unified framework for responding to the wide range of attacks malicious insiders and outsiders can launch.*

6. **Architecture for an Artificial Immune System** [20]
   Abstract: *An artificial immune system (ARTIS) is described which incorporates many properties of natural immune systems, including diversity, distributed computation, error tolerance, dynamic learning and adaptation and self-monitoring. ARTIS is a general framework for a distributed adaptive system and could, in principle, be applied to many domains. In this paper, ARTIS is applied to computer security, in the form of a network intrusion detection system called LISYS. LISYS is described and shown to be effective at detecting intrusions, while maintaining low false positive rates. Finally, similarities and differences between ARTIS and Holland's classifier systems are discussed.*

7. **Inoculating Software for Survivability** [19]
   Abstract: *In early 1998, several dozen computer systems in U.S. military installations and government facilities were successfully hacked, resulting in a full-scale Defense Department response now known as Operation Solar Sunrise. The attacks successfully broke into systems belonging to the Navy and Air Force as well as to federally funded research laboratories including Oak Ridge National Laboratory, Brookhaven National Laboratories, U.C. Berkeley, and MIT. Although no classified systems were allegedly compromised, the attackers were able to obtain system privileges that could be used to read password files, delete files, or create back doors for later re-entry. Despite being called the most organized and systematic attack to date against the Department of Defense*

systems by the U.S. Deputy Defense Secretary, these attacks were not the work of an organized terrorist group or nation; rather, authorities believe two northern California teenagers under the tutelage of an Israeli computer hacker were responsible for breaking into these systems, simply because they could.

8. **Crash-Only Software** [12]
   Abstract: *Crash-only programs crash safely and recover quickly. There is only one way to stop such software – by crashing it – and only one way to bring it up – by initiating recovery. Crash-only systems are built from crash-only components, and the use of transparent component-level retries hides intra-system component crashes from end users. In this paper we advocate a crash-only design for Internet systems, showing that it can lead to more reliable, predictable code and faster, more effective recovery. We present ideas on how to build such crash-only Internet services, taking successful techniques to their logical extreme.*

9. **Building Diverse Computer Systems** [16]
   Abstract: *Diversity is an important source of robustness in biological systems. Computers, by contrast, are notable for their lack of diversity. Although homogeneous systems have many advantages, the beneficial effects of diversity in computing systems have been overlooked, specifically in the area of computer security. Several methods of achieving software diversity are discussed based on randomization that respect the specified behavior of the program. Such randomization could potentially increase the robustness of software systems with minimal impact on convenience, usability, and efficiency. Randomization of the amount of memory allocated on a stack frame is show to disrupt a simple buffer overflow attack.*

10. **Feedback Control Applied to Survivability: A Host-Based Autonomic Defense System** [26]
    Abstract: *We address the problem of information system survivability, or dynamically preserving intended functionality and computational performance, in the face of malicious intrusive activity. A feedback control approach is proposed that enables trade-offs between the failure cost of a compromised information system and the maintenance cost of ongoing defensive countermeasures. Online implementation features an inexpensive computation architecture consisting of a sensor-driven recursive estimator followed by an estimate-driven response selector. Offline design features a systematic empirical procedure utilizing a suite of mathematical modeling and numerical optimization tools. The engineering challenge is to generate domain models and decision strategies offline via tractable methods with achieving online effectiveness. We illustrate the approach with experimentation results for a prototype autonomic defense system that protects its host, a Linux-based web-server, against an automated Internet worm attack. The overall approach applies to other types of computer attacks, network-level security and other domains that could benefit from automatic decision-making based on a sequence of sensor measurements.*

4.2 On Host Defense Systems

1. **Improving Host Security with System Call Policies** [36]
   Abstract: *We introduce a system that eliminates the need to run programs in privileged process contexts. Using our system, programs run unprivileged but may execute certain operations with elevated privileges as determined by a configurable policy eliminating the need for suid or sgid binaries. We present the design and analysis of the Systrace facility which supports fine grained process confinement, intrusion detection, auditing and privilege elevation. It also facilitates the often difficult process of policy generation. With Systrace, it is possible to generate policies automatically in a training session or generate them interactively during program execution. The policies describe the desired behavior of services or user applications on a system call level and are enforced to prevent operations that are not explicitly permitted. We show that Systrace is efficient and does not impose significant performance penalties.*

2. **Automated Response Using System-Call Delays** [46]
   Abstract: *Automated intrusion response is an important unsolved problem in computer security. A system called pH (for process homeostasis) is described which can successfully detect and stop intrusions before the target system is compromised. In its current form, pH monitors every executing process on a computer at the system-call level, and responds to anomalies by either delaying or aborting system calls. The paper presents the rationale for pH, its design and implementation, and a set of initial experimental results.*

3. **Using Specification-Based Intrusion Detection for Automated Response** [4]
   Abstract: *One of the most controversial issues in intrusion detection is automating responses to intrusions, which can provide a more efficient, quicker, and precise way to react to an attack in progress than a human. However, it comes with several disadvantages that can lead to a waste of resources, which has so far prevented wide acceptance of automated response-enabled systems. We feel that a structured approach to the problem is needed that will account for the above mentioned disadvantages. In this work, we briefly describe what has been done in the area before. Then we start addressing the problem by coupling automated response with specification-based, host-based intrusion detection. We describe the system map, and the map-based action cost model that give us the basis for deciding on response strategy. We also show the process of suspending the attack, and designing the optimal response strategy, even in the presence of uncertainty. Finally, we discuss the implementation issues, our experience with the early automated response agent prototype, the Automated Response Broker (ARB), and suggest topics for further research.*

4. **Operating System Stability and Security through Process Homeostasis** [48]
   Abstract: *Modern computer systems are plagued with stability and security problems: applications lose data, web servers are hacked, and systems crash under heavy load. Many of these problems arise from rare program behaviors. pH (process Homeostasis) is a Linux 2.2 kernel extension which detects unusual program behavior and responds by slowing down that behavior. Inspired by the homeostatic mechanisms organisms use to stabilize their internal environment, pH detects changes in program behavior by observing changes in short sequences of system calls. When pH determines that a process is behaving unusually, it responds by slowing down that process's system calls. If the anomaly corresponds to a security violation, delays often stop attacks before they can do damage. Delays also give users time to decide whether further actions are warranted.*
   My dissertation describes the rationale, design, and behavior of pH. Experimental results are reported which show that pH effectively captures the normal behavior of a variety of programs under normal use conditions. This captured behavior allows it to detect anomalies with a low rate of false positives (as low as 1 user intervention every five days). Data are presented that show pH responds effectively and autonomously to buffer overflows, trojan code, and kernel security flaws. pH can also help administrators by detecting newly-introduced configuration errors. At the same time, pH is extremely lightweight: it incurs a general performance penalty of only a few percent, a slowdown that is imperceptible in practice. The pH prototype is licensed under the GNU General Public License and is available for download at `http://www.cs.unm.edu/~soma/pH/`.

5. **Automatic Data Structure Repair for Self-Healing Services** [14]
   Abstract: *We have developed a system that accepts a specification of key data structure constraints, then dynamically detects and repairs violations of these constraints, enabling the program to recover from otherwise crippling errors to continue to execute productively. We present our experience using our system to repair violated constraints in a simplified version of the ext2 file system and in the CTAS air-traffic control program. Our experience indicates that the specifications are relatively straightforward to develop and that our technique enables the applications to effectively recover from data structure corruption errors.*

6. **Enhancing Server Availability and Security Through Failure-Oblivious Computing** [40]
   Abstract: *We present a new technique,* failure-oblivious computing*, that enables servers to execute through memory errors without memory corruption. Our safe compiler for C inserts checks that dynamically detect invalid memory accesses. Instead of terminating or throwing an exception, the generated code simply discards invalid writes and manufactures values to return for invalid reads, enabling the server to continue its normal execution path.*
   We have applied failure-oblivious computing to a set of widely-used servers from the Linux-based open source computing environment. Our results show that our techniques 1) make these servers invulnerable to known security attacks that exploit memory errors, and 2) enable the servers to continue to operate successfully to service legitimate requests and satisfy the needs of their users even after attacks trigger their memory errors.
   We have observed several reasons for this successful continued execution. When the memory errors occur in irrelevant computations, failure-oblivious computing enables the server to execute through the memory errors to continue on to execute the relevant computation. Even when the memory errors occur in relevant computations, failure-oblivious computing converts requests that trigger unanticipated and dangerous execution paths into anticipated invalid inputs, which the error-handling logic in the server rejects. Because servers tend to have small error propagation distances (localized errors in the computation for one request tend to have little or no effect

on the computations for subsequent requests), redirecting reads that would otherwise cause addressing errors and discarding writes that would otherwise corrupt critical data structures (such as the call stack) localizes the effect of the memory errors, prevents addressing exceptions from terminating the computation, and enables the server to continue on to successfully process subsequent requests. The overall result is a substantial extension of the range of request that the server can successfully process.

7. **Continual Repair for Windows Using the Event Log** [38]
   Abstract: *There is good reason to base intrusion detection on data from the host. Unfortunately, most operating systems do not provide all the data needed in readily available logs. Ironically, perhaps, Windows NT and its successor, Windows 2000, provide much of the necessary data, at least for security events. We have developed a host-based intrusion detector for these platforms that meets the generally accepted criteria for a good Intrusion Detection System. Its architecture is sufficiently flexible to meet these criteria largely by relying on native mechanisms. Where there are identified gaps in the data from the native security event log, they can be filled by data from other sensors by using the same event logging interface. The IDS will also terminate unauthorized processes, delete unauthorized files, and restore deleted or modified files continually without lengthy recover due to compromise. We call this feature Continual Repair. It is an existence proof that self-regenerative systems are possible.*

8. **Secure Execution Via Program Shepherding** [24]
   Abstract: *We introduce* program shepherding*, a method for monitoring control flow transfers during program execution to enforce a security policy. Program shepherding provides three techniques as building blocks for security policies. First, shepherding can restrict execution privileges on the basis of code origins. This distinction can ensure that malicious code masquerading as data is never executed, thwarting a large class of security attacks. Second, shepherding can restrict control transfers based on instruction class, source, and target. For example, shepherding can forbid execution of shared library code except through declared entry points, and can ensure that a return instruction only targets the instruction after a call. Finally, shepherding guarantees that sandboxing checks placed around any type of program operation will never be bypassed. We have implemented these capabilities efficiently in a runtime system with minimal or no performance penalties. This system operates on unmodified native binaries, requires no special hardware or operating system support, and runs on existing IA-32 machines under both Linux and Windows.*

9. **Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits** [50]
   Abstract: *Software patching has not been effective as a first-line defense against large-scale worm attacks, even when patches have long been available for their corresponding vulnerabilities. Generally, people have been reluctant to patch their systems immediately, because patches are perceived to be unreliable and disruptive to apply. To address this problem, we propose a first-line worm defense in the network stack, using* shields *– vulnerability-specific, exploit-generic network filters installed in end systems once a vulnerability is discovered, but before a path is applied. These filters examine the incoming or outgoing traffic of vulnerable applications, and correct traffic that exploits vulnerabilities. Shields are less disruptive to install and uninstall, easier to test for bad side effects, and hence more reliable than traditional software patches. Further, shields are resilient to polymorphic or metamorphic variations of exploits.*
   In this paper, we show that this concept is feasible by describing a prototype Shield framework implementation that filters traffic above the transport layer. We have designed a safe and restrictive language to describe vulnerabilities as partial state machines of the vulnerable application. The expressiveness of the language has been verified by encoding the signatures of several known vulnerabilities. Our evaluation provides evidence of Shield's low false positive rate and small impact on application throughput. An examination of a sample set of known vulnerabilities suggests that Shield could be used to prevent exploitation of a substantial fraction of the most dangerous ones.

10. **AngeL: a tool to disarm computer systems** [11]
    Abstract: *In this paper we present a tool designed to intercept attacks at the host where they are launched so as to block them before they reach their targets. The tool works both for attacks targeted on the local host and on hosts connected to the network. In the current implementation it can detect and block more than 70 attacks as reported in the literature.*
    The tool is based on the idea of improving the overall security of the Internet by connecting disarmed systems,

i.e., hosts that cannot launch attacks against other hosts. Such a strategy was presented in [prior work]. Here we present an extended version of the tool that has been engineered to consider a wide variety of attacks and to run on various releases of the Linux kernel and the experience learned in building such a tool. A protection mechanism of the tool itself that prevents its removal is also implemented. Experimental results of the impact of the tool on system performance show that the overhead introduced by the tool is negligible from the user's perspective, thus it is not expected to be a hindrance to the successful deployment of the tool.

11. **Access Control Based on Execution History** [2]
Abstract: *Security is a major, frequent concern in extensible software systems such as Java Virtual Machines and the Common Language Runtime. These systems aim to enable simple, classic applets and also, for example, distributed applications, Web services, and programmable networks, with appropriate security expectations. Accordingly, they feature elaborate constructs and mechanisms for associating rights with code, including a technique for determining the run-time rights of a piece of code as a function of the state of the execution stack. These mechanisms prevent many security holes, but they are inherently partial and they have proved difficult to use reliably.*
We motivate and describe a new model for assigning rights to code: in short, the run-time rights of a piece of code are determined by examining hte attributes of any pieces of code that have run (including their origins) and any explicit requests to augment rights. This history-based model addresses security concerns while avoiding pitfalls. We analyze the model in detail; in particular, we discuss its relation to the stack-based model and to the policies and mechanisms of underlying operating systems, and we consider implementation techniques. In support of the model, we also introduce and implement high-level constructs for security, which should be incorporated in libraries or (even better) in programming languages.

4.3 On Network Defense Systems

1. **A Network Worm Vaccine Architecture** [45]
Abstract: *The ability of worms to spread at rates that effectively preclude human-directed reaction has elevated them to a first-class security threat to distributed systems. We present the first reaction mechanism that seeks to automatically patch vulnerable software. Our system employs a collection of sensors that detect and capture potential worm infection vectors. We automatically test the effects of these vectors on appropriately-instrumented sandboxed instances of the targeted application, trying to identify the exploited software weakness. Our heuristics allow us to automatically generate patches that can protect against certain classes of attack, and test the resistance of the patched application against the infection vector. We describe our system architecture, discuss the various components, and propose directions for future research.*

2. **An Automated Defense System to Counter Internet Worms** [42]
Abstract: *Our society is highly dependent on network services such as the Web, email, and collaborative P2P enterprise applications. But what if such infrastructures were suddenly torn down? Both past incidents and research studies show that a well-engineered Internet worm can accomplish such a task in a fairly simple way and, most notably, in a matter of a few minutes. This clearly rules out the possibility of manually countering worm outbreaks. We present a testbed that operates on a cluster of computers and emulates very large networks for purposes of experimentation. A wide variety of worm properties can be studied and network topologies of interest constructed. A reactive control system, based on the Willow architecture, operates on top of the testbed and provides a monitor/analyze/respond approach to deal with infections automatically. The logic driving the control system is synthesized from a formal specification, which is based on control rules that correlate sensor events. Details of our highly configurable testbed, the theory of operation of the Willow architecture, the features of the specification language, and various experimental performance results are presented.*

3. **A Hybrid Quarantine Defense** [34]
Abstract: *We study the strengths, weaknesses, and potential synergies of two complementary worm quarantine defense strategies under various worm attack profiles. We observe their abilities to delay or suppress infection growth rates under two propagation techniques and three scan rates, and explore the quarantine strategies. We compare the performance of the individual strategies against a hybrid combination strategy, and conclude that the hybrid strategy yields substantial performance improvements, beyond what either technique provides independently. This result offers potential new directions in hybrid quarantine defenses.*

4. **On Achieving Software Diversity for Improved Network Security Using Distributed Coloring Algorithms** [30]
   Abstract: *It is widely believed that diversity in operating systems, software packages, and hardware platforms will decrease the virulence of worms and the effectiveness of repeated applications of single attacks. Research efforts in the field have focused on introducing diversity using a variety of techniques on a system-by-system basis. This paper, on the other hand, assumes the availability of diverse software packages for each system and then seeks to increase the intrinsic value of available diversity by considering the entire computer network. We present several distributed algorithms for the assignment of distinct software packages to individual systems and analyze their performance. Our goal is to limit the ability of a malicious node to use a single attack to compromise its neighboring nodes, and by extension, the rest of the nodes in the network. The algorithms themselves are analyzed for attack tolerance, and strategies for improving the security of the individual software assignment schemes are presented. We present a comparative analysis of our algorithms using simulation results on a topology obtained from e-mail traffic logs between users at our institution. We find that hybrid version of our algorithms incorporating multiple assignment strategies achieve better attack tolerance than any given assignment strategy. Our work thus shows that diversity must be introduced at all levels of system design, including any scheme that is used to introduce diversity itself.*

5. **Implementing Pushback: Router-Based Defense Against DDoS Attacks** [21]
   Abstract: *Pushback is a mechanism for defending against distributed denial-of-service (DDoS) attacks. DDoS attacks are treated as a congestion-control problem, but because most such congestion is caused by malicious hosts not obeying traditional end-to-end congestion control, the problem must be handled by the routers. Functionality is added to each router to detect and preferentially drop packets that probably belong to an attack. Upstream routers are also notified to drop such packets (hence the term* Pushback*) in order that the router's resources be used to route legitimate traffic. In this paper we present an architecture for Pushback, its implementation under FreeBSD, and suggestions for how such a system can be implemented in core routers.*

6. **Tracing Based Active Intrusion Response** [54]
   Abstract: *Network-based intrusion has become a serious threat to today s highly networked information systems, existing intrusion defense approaches such as intrusion prevention, detection, tolerance and response are passive in response to network-based intrusions in that their countermeasures are limited to being local to the intrusion target and there is no automated, network-wide counteraction against detected intrusions. While they all play an important role in counteracting network-based intrusion, they do not, however, effectively address the root cause of the problem – intruders.*
   What missing from existing intrusion prevention, detection, tolerance and response is an effective way to identify network-based intruders and hold them accountable for their intrusions. Network-based intrusion can not be effectively repelled or eliminated until its source is known.
   In this paper, we propose Tracing Based Active Intrusion Response (TBAIR) as a new way to address the problem of network-based intrusion. Based on Sleepy Watermark Tracing (SWT), TBAIR is able to effectively trace the detected intrusion that utilizes stepping stone to disguise its origin at real-time, and dynamically push the intrusion countermeasures such as remote monitoring, blocking, containment and isolation close to the source of the intrusion. Through SWT s unique active watermark technique, TBAIR is able to trace even when the intrusion connection is idle. Therefore it helps to apprehend the intruders on the spot and hold them accountable for their intrusions.

7. **Dynamic Access Control: Preserving Safety and Trust for Network Defense Operations** [28]
   Abstract: *We investigate the cost of changing access control policies dynamically as a response action in computer network defense. We compare and contrast the use of access lists and capability lists in this regard, and develop a quantitative feel for the performance overheads and storage requirements. We also explore the issues related to preserving safety properties and trust assumptions during this process. We suggest augmentations to policy specifications that can guarantee these properties in spite of dynamic changes to system state. Using the lessons learned from this exercise, we apply these techniques in the design of dynamic access controls for dynamic environments.*

8. **Anomalous Payload-based Network Intrusion Detection** [51]
   Abstract: *We present a payload-based anomaly detector, we call PAYL, for intrusion detection. PAYL models*

the normal application payload of network traffic in a fully automatic, unsupervised and very efficient fashion. We first compute during a training phase a profile byte frequency distribution and their standard deviation of the application payload flowing to a single host and port. We then use Mahalanobis distance during the detection phase to calculate the similarity of new data against the pre-computed profile. The detector compares this measure against a threshold and generates an alert when the distance of the new input exceeds this threshold. We demonstrate the surprising effectiveness of the method on the 1999 DARPA IDS dataset and a live dataset we collected on the Columbia CS department network. In once case nearly 100% accuracy is achieved with 0.1% false positive rate for port 80 traffic.

9. **Adaptive Use of Network-Centric Mechanisms in Cyber-Defense** [1]
   Abstract: *Attacks against distributed systems frequently start at the network layer by gathering network related information (such as open TCP ports) and continue on by exhausting resources, or abusing protocols. Defending against network-based attacks is a major focus area in the APOD (Applications that Participate in Their Own Defense) project, which set out to develop technologies that increase an application's resilience against cyber attacks. This paper gives an overview of APOD's current set of network-level defenses. Specific network-based defense mechanisms are described first, followed by a discussion on how to use them in local defensive behavior. Defense strategies, which specify coordinated defensive behavior across a distributed system, are discussed next, followed by results from initial experimental evaluation.*

10. **On-Line Intrusion Detection and Attack Prevention Using Diversity, Generate-and-Test, and Generalization** [39]
    Abstract: *We have built a system for protecting Internet services to securely connected, known users. It implements a generate-and-test approach for on-line attack identification and uses similarity rules for generalization of attack signatures. We can immediately protect the system from many variants of previously unknown attacks without debilitating waits for anti-virus updates or software patches. Unique to our approach is the use of diverse process pairs not only for isolation benefits but also for detection. The architecture uses the comparison of outputs from diverse applications to provide a significant and novel intrusion detection capability. With this technique, we gain the benefits of n-version programming without its controversial disadvantages. The isolation of intrusions is mainly achieved with an out-of-band control system that separates the primary and backup systems. It also initiates attack diagnosis and blocking, and recovery, which is accelerated by continual repair.*

4.4 Other Work

These papers are not part of the candidacy list, but provide additional background and support material for understanding the context of the papers presented as part of the exam. They are listed here for completeness.

4.4.1 Background on Automated Response and Intrusion Tolerance

1. **`http://www.securityfocus.com/infocus/1540`**
   Summary: *This short technical article covers two ways that naive response mechanisms can be frustrated and subverted to become counter productive.*

2. **Strike Back: Offensive Actions in Information Warfare** [52]
   Abstract:

3. **Intrusion-detection for incident-response, using a military battlefield-intelligence process** [**?**]
   Abstract:

4. **Intrusion Tolerant Systems** [33]
   Abstract: *A position paper.*

5. **Intrusion Detection, Diagnosis, and Recovery with Self-Securing Storage** [49]
   Abstract: *Self-securing storage turns storage devices into active parts of an intrusion survival strategy. From behind a thin storage interface (e.g., SCSI or CIFS), a self-securing storage server can watch storage requests, keep a record of all storage activity, and prevent compromised clients from destroying stored data. This paper describes three ways self securing storage enhances an administrator's ability to detect, diagnose, and recover*

from client system intrusions. First, storage-based intrusion detection offers a new observation point for noticing suspect activity. Second, post-hoc intrusion diagnosis starts with a plethora of normally-unavailable information. Finally, post-intrusion recovery is reduced to restarting the system with a pre-intrusion storage image retained by the server. Combined, these features can improve an organization s ability to survive successful digital intrusions.

6. **NIDAR: The Design and Implementation of an Intrusion Detection System** `http://www.raid-symposium.org/raid98/Prog_RAID98/Talks.html#Hwee_10` *This abstract briefly mentions response capabilities but does not elaborate on them.*

7. **Micael: An Autonomous Mobile Agent System to Protect New Generation Networked Applications** [13]
Abstract:

8. **Survival by Defense-Enabling** [32]
Abstract: *Attack survival, which means the ability to provide some level of service despite an ongoing attack by tolerating its impact, is an important objective of security research. In this paper we present a new approach to survivability and intrusion tolerance. Our approach, which we call "survival by defense", is based on the observation that many applications can be given increased resistance to malicious attack even though the environment in which they run is untrustworthy. This paper describes the concept of "survival by defense" in general and explains the assumptions on which it depends. We will also explain the goals of survival by defense and how they can be achieved.*

4.4.2 Computer Immunology

1. **Internet Quarantine: Requirements for Containing Self-Propagating Code** [15]
Abstract:

2. **A Hybrid IDS Architecture Based on the Immune System** [37]
Abstract: *The human immune system provides a rich source of inspiration for computer network security. Exploring this analogy the authors propose a hybrid intrusion detection architecture that has the same learning and adaptive capability of the human immune system.*

3. **Principles of a Computer Immune System** [47]
Abstract: *Natural immune systems provide a rich source of inspiration for computer security in the age of the Internet. Immune systems have many features that are desirable for the imperfect, uncontrolled, and open environments in which most computers currently exist. These include distributability, diversity, disposability, adaptability, autonomy, dynamic coverage, anomaly detection, multiple layers, identity via behavior, no trusted components, and imperfect detection. These principles suggest a wide variety of architectures for a computer immune system.*

4. **Computer Immunology** [17]
Abstract: *Natural immune systems protect animals from dangerous foreign pathogens, including bacteria, viruses, parasites, and toxins. Their role in the body is analogous to that of computer security systems in computing. Although there are many differences between living organisms and computer systems, this article argues that the similarities are compelling and could point the way to improved computer security. Improvements can be achieved by designing computer immune systems that have some of the important properties illustrated by natural immune systems. These include multi-layered protection, highly distributed detection and memory systems, diversity of detection ability across individuals, inexact matching strategies, and sensitivity to most new foreign patterns. We first give an overview of how the immune system relates to computer security. We then illustrate these ideas with two examples.*

5. **A Cooperative Immunization System for an Untrusting Internet** [3]
Abstract:

6. **Cooperative Response Strategies for Large Scale Attack Mitigation** [29]
Abstract:

4.4.3 System Call Interposition and Sandboxes

1. **Hardening COTS Software with Generic Software Wrappers** [18]
   Abstract: *Numerous techniques exist to augment the security functionality of Commercial Off-the-Shelf (COTS) applications and operating systems, making them more suitable for use in mission-critical systems. Although individually useful, as a group these techniques present difficulties to system developers because they are not based on a common framework which might simplify integration and promote portability and reuse. This paper presents techniques for developing Generic Software Wrappers - protected, non-bypassable kernel-resident software extensions for augmenting security without modification of COTS source. We describe the key elements of our work: our high-level Wrapper Definition Language (WDL), and our framework for configuring, activating, and managing wrappers. We also discuss code reuse, automatic management of extensions, a framework for system-building through composition, platform-independence, and our experiences with our Solaris and FreeBSD prototypes.*

2. **Traps and Pitfalls: Practical Problems in System Call Interposition Based Security Tools**
   `http://www.stanford.edu/~talg/papers/traps/traps-ndss03.pdf`

3. **Synthesizing Fast Intrusion Prevention/Detection Systems from High-Level Specifications** [43]
   Abstract: *To build survivable information systems (i.e., systems that continue to provide their services in spite of coordinated attacks), it is necessary to detect and isolate intrusions before they impact system performance or functionality. Previous research in this area has focussed primarily on detecting intrusions after the fact, rather than preventing them in the first place. We have developed a new approach based on specifying intended program behaviors using patterns over sequences of system calls. The patterns can also capture conditions on the values of system-call arguments. At runtime, we intercept the system calls made by processes, compare them against specifications, and disallow (or otherwise modify) those calls that deviate from specifications. Since our approach is capable of modifying a system call before it is delivered to the operating system kernel, it is capable of reacting before any damage-causing system call is executed by a process under attack. We present our specification language and illustrate its use by developing a specification for the ftp server. Observe that in our approach, every system call is intercepted and subject to potentially expensive operations for matching against many patterns that specify normal/abnormal behavior. Thus, minimizing the overheads incurred for pattern-matching is critical for the viability of our approach. We solve this problem by developing a new, low-overhead algorithm for matching runtime behaviors against specifications. A salient feature of our algorithm is that its runtime is almost independent of the number of patterns. In most cases, it uses a constant amount of time per system call intercepted, and uses a constant amount of storage, both independent of either the size or number of patterns. These benefits make our algorithm useful for many other intrusion detection methods that employ pattern-matching. We describe our algorithm, and evaluate its performance through experiments.*

4. **Detecting and Countering System Intrusions Using Software Wrappers** [25]
   Abstract: *This paper introduces an approach that integrates intrusion detection (ID) techniques with software wrapping technology to enhance a system's ability to defend against intrusions. In particular, we employ the NAI Labs Generic Software Wrapper Toolkit to implement all or part of an intrusion detection system as ID wrappers. An ID wrapper is a software layer dynamically inserted into the kernel that can selectively intercept and analyze system calls performed by processes as well as respond to intrusive events. We have implemented several ID wrappers that employ three different major intrusion detection techniques. Also, we have combined different ID techniques by composing ID wrappers at run-time. We tested the individual and composed ID wrappers using several existing attacks and measured their impact on observed application performance. We conclude that intrusion detection algorithms can be easily encoded as wrappers that perform efficiently inside the kernel. Also, kernel-resident ID wrappers can be easily managed, allowing cooperation among multiple combined techniques to enforce a coherent global ID policy. In addition, intrusion detection algorithms can benefit from the extra data made accessible by wrappers.*

5. **Operating System Enhacements to Prevent the Misuse of System Calls** [8]
   Abstract:

4.4.4 Misc

1. **Transparent Run-Time Defense Against Stack Smashing Attacks** [6]
   Abstract: *The exploitation of buffer overflow vulnerabilities in process stacks constitutes a significant portion of security attacks. We present two new methods to detect and handle such attacks. In contrast to previous work, the new methods work with any existing pre-compiled executable and can be used transparently per-process as well as on a system-wide basis. The first method intercepts all calls to library functions known to be vulnerable. A substitute version of the corresponding function implements the original functionality, but in a manner that ensures that any buffer overflows are contained within the current stack frame. The second method uses binary modification of the process memory to force verification of critical elements of stacks before use. We have implemented both methods on Linux as dynamically loadable libraries and shown that both libraries detect several known attacks. The performance overhead of these libraries range from negligible to 15%.*

2. **A Binary Rewriting Defense Against Stack based Buffer Overflow Attacks** [35]
   Abstract: *Buffer overflow attack is the most common and arguably the most dangerous attack method used in Internet security breach incidents reported in the public literature. Various solutions have been developed to address the buffer vulnerability problem in both research and commercial communities. Almost all the solutions that provide adequate protection against buffer overflow attacks are implemented as compiler extensions and hence require the source code of the programs being protected to be available so that they can be re-compiled. While this requirement is reasonable in many cases, there are scenarios in which it is not feasible, e.g., legacy applications that are purchased from an outside vendor. The work reported in this paper explores the application of static binary translation to protect Internet software from buffer overflow attacks. Specifically, we use a binary rewriting approach to augment existing Win32/Intel Portable Executable (PE) binary programs with a return address defense (RAD) mechanisms, which protects the integrity of the return address on the stack with a redundant copy. [...]*

3. **Bend, Don't Break: Using Reconfiguration to Achieve Survivability** [53]
   Abstract: *...We are designing a secure, automated framework for proactive and reactive reconfiguration of large-scale, heterogeneous, distributed systems so that critical networked computing enterprises can tolerate intrusions and continue to provide an acceptable level of service. Proactive reconfiguration adds, removes, and replaces components and interconnections to cause a system to assume postures that achieve enterprise-wide intrusion tolerance goals, such as increased resilience to specific kinds of attacks or increased preparedness for recovery from specific kinds of failures...*

4. **Address Obfuscation: An Efficient Approach to Combat a Broad Range of Memory Error Exploits** [9]
   Abstract: *Attacks which exploit memory programming errors (such as buffer overflows) are one of today's most serious security threats. These attacks require an attacker to have an in-depth understanding of the internal details of a victim program, including the locations of critical data and/or code.* Program obfuscation *is a general technique for securing programs by making it difficult for attackers to acquire such a detailed understanding. This paper develops a systematic study of a particular kind of obfuscation called address obfuscation that randomizes the location of victim program data and code. We discuss different implementation strategies to randomize the absolute locations of data and code, as well as relative distances between data locations. We then present our implementation that transforms object files and executables at link time and load time. [...]*

5. **On the Effectiveness of Address-Space Randomization** [44]
   Abstract: *Address-space randomization is a technique used to fortify systems against buffer overflow attacks. The idea is to introduce artificial diversity by randomizing the memory location of certain system components. This mechanism is available for both Linux (via PaX ASLR) and OpenBSD. We study the effectiveness of address-space randomization and find that its utility on 32-bit architectures is limited by the number of bits available for address randomization. In particular, we demonstrate a* derandomization attack *that will convert any standard buffer-overflow exploit into an exploit that works against systems protected by address-space randomization. The resulting exploit is as effective as the original exploit, although it takes a little longer to compromise a target machine: on average 216 seconds to compromise Apache running on a Linux PaX ASLR system. The attack does not require running code on the stack.*
   We also explore various ways of strengthening address-space randomization and point out weaknesses in each.

Surprisingly, increasing the frequency of re-randomizations adds at most 1 bit of security. Furthermore, compile-time randomization appears to be more effective than runtime randomization. We conclude that, on 32-bit architectures, the only benefit of PaX-like address-space randomization is a small slowdown in worm propagation speed. The cost of randomization is extra complexity in system support.

6. **Countering Code-Injection Attacks With Instruction-Set Randomization** [22]
   Abstract: *We describe a new, general approach for safeguarding systems against any type of code-injection attack. We apply Kerckhoff's principle, by creating process-specific randomized instruction sets (e.g., machine instructions) of the system executing potentially vulnerable software. An attacker who does not know the key to the randomization algorithm will inject code that is invalid for that randomized processor, causing a runtime exception. To determine the difficulty of integrating support for the proposed mechanism in the operating system, we modified the Linux kernel, the GNU binutils program, and the bochs-x86 emulator. Although the performance penalty is significant, our prototype demonstrates the feasibility of the approach, and should be directly usable on a suitable-modified processor (e.g., the Transmeta Crusoe).*
   Our approach is equally applicable against code-injecting attacks in scripting and interpreted languages, e.g., web-based SQL injection. We demonstrate this by modifying the Perl interpreter to permit randomized script execution. The performance penalty in this case is minimal. Where our proposed approach is feasible (i.e., in an emulated environment, in the presence of programmable or specialized hardware, or in interpreted languages), it can serve as a low-overhead protection mechanism, and can easily complement other mechanisms.

7. **Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks** [7]
   Abstract: *Binary code injection into an executing program is a common form of attack. Most current defenses against this form of attack use a 'guard all doors' strategy, trying to block the avenues by which execution can be diverted. We describe a complementary method of protection, which disrupts foreign code execution regardless of how the code is injected. A unique and private machine instruction set for each executing program would make it difficult for an outsider to design binary attack code against that program and impossible to use the same binary attack code against multiple machines. As a proof of concept, we describe a* randomized instruction set emulator *(RISE), based on the open-source Valgrind x86-to-x86 binary translator. The prototype disrupts binary code injection attacks against a program without requiring its recompilation, linking, or access to source code. The paper describes the RISE implementation and its limitations, gives evidence demonstrating that RISE defeats common attacks, considers how the dense x86 instruction set affects the method, and discusses potential extensions of the idea.*

8. **SQLRand: Preventing SQL Injection Attacks** [10]
   Abstract: *We present a practical protection mechanism against SQL injection attacks. Such attacks target databases that are accessible through a web frontend, and take advantage of flaws in the input validation logic of Web components such as CGI scripts. We apply the concept of instruction-set randomization to SQL, creating instances of the language that are unpredictable to the attacker. Queries injected by the attacker will be caught and terminated by the database parser. We show how to use this technique with the MySQL database using an intermediary proxy that translates the random SQL to its standard language. Our mechanism imposes negligible performance overhead to query processing and can be easily retrofitted to existing systems.*

5 Reading List

References

[1] Adaptive Use of Network-Centric Mechanisms in Cyber-Defense. In *Proceedings of the 2nd IEEE International Symposium on Network Computing and Applications*, April 2003.

[2] Martin Abadi and Cedric Fournet. Access Control Based on Execution History. In *Proceedings of the 2003 Symposium on Network and Distributed Systems Security (NDSS)*, 2003.

[3] Kostas Anagnostakis, Michael B. Greenwald, Sotiris Ioannidis, Angelos D. Keromytis, and Dekai Li. A Cooperative Immunization System for an Untrusting Internet. In *Proceedings of the 11th IEEE International Conference on Networks (ICON)*, pages 403–408, October 2003.

[4] Ivan Balepin, Sergei Maltsev, Jeff Rowe, and Karl Levitt. Using Specification-Based Intrusion Detection for Automated Response. In *Proceedings of the 6th International Workshop on Recent Advances in Intrusion Detection (RAID)*, September 2003.

[5] Boaz Barak, Amir Herzberg, Dalit Naor, and Eldad Shai. The Proactive Security Toolkit and Applications. In *Proceedings of the 6th ACM Conference on Computer and Communications Security (CCS)*, November 1999.

[6] Arash Baratloo, Navjot Singh, and Timothy Tsai. Transparent Run-Time Defense Against Stack Smashing Attacks. In *Proceedings of the USENIX General Technical Conference*, 2000.

[7] E. G. Barrantes, D. H. Ackley, S. Forrest, T. S. Palmer, D. Stefanovic, and D. D. Zovi. Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, pages 281–289, October 2003.

[8] M. Bernaschi, L. Mancini, and E. Gabrielli. Operating System Enhancements to Prevent the Misuse of System Calls. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS)*, page 174, 2000.

[9] S. Bhatkar, D.C. DuVarney, and R. Sekar. Address Obfuscation: An Efficient Approach to Combat a Broad Range of Memory Error Exploits. In *Proceedings of the 12th USENIX Security Symposium*, pages 105–120, August 2003.

[10] Stephen Boyd and Angelos Keromytis. SQLrand: Preventing SQL Injection Attacks. In *Proceedings of the 2nd Applied Cryptography and Network Security (ACNS) Conference*, June 2004.

[11] Danilo Bruschi and Emilia Rosti. AngeL: a tool to disarm computer systems. In *Proceedings of the New Security Paradigms Workshop (NSPW)*, pages 63–69, September 2001.

[12] George Candea and Armando Fox. Crash-only software. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems*, May 2003.

[13] Jose Duarte de Queiroz et al. Micael: An Autonomous Mobile Agent System to Protect New Generation Networked Applications. In *Proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection (RAID)*, 1999.

[14] Brian Demsky and Martin C. Rinard. Automatic Data Structure Repair for Self-Healing Systems. In *Proceedings of the 1st Workshop on Algorithms and Architectures for Self-Managing Systems*, June 2003.

[15] Moore et al. Internet Quarantine: Requirements for Containing Self-Propagating Code. In *XXX*, 2003.

[16] S. Forrest, A. Somayaji, and D. Ackley. Building Diverse Computer Systems. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems*, pages 67–72, 1997.

[17] Stephanie Forrest, Steven A. Hofmeyr, and Anil Somayaji. Computer Immunology. *Communications of the ACM*, 40(10):88–96, 1997.

[18] Timothy Fraser, Lee Badger, and Mark Feldman. Hardening COTS Software with Generic Software Wrappers. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, 1999.

[19] Anup K. Ghosh and Jeffery M. Voas. Innoculating Software for Survivability. *Communications of the ACM*, 42(7), 1999.

[20] S. Hofmeyr and S. Forrest. Architecture for an Artificial Immune System. *Evolutionary Computation Journal*, 8(4):443–473, 2000.

[21] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proceedings of Network and Distributed System Security (NDSS) Symposium*, 1775 Wiehle Ave., Suite 102, Reston, VA 20190, February 2002. The Internet Society.

[22] Gaurav S. Kc, Angelos D. Keromytis, and Vassilis Prevelakis. Countering Code-Injection Attacks With Instruction-Set Randomization. In *Proceedings of the ACM Computer and Communications Security (CCS) Conference*, pages 272–280, October 2003.

[23] Angelos D. Keromytis, Janak Parekh, Philip N. Gross, Gail Kaiser, Vishal Misra, Jason Nieh, Dan Rubenstein, and Sal Stolfo. A Holistic Approach to Service Survivability. In *Proceedings of the 1st ACM Workshop on Survivable and Self-Regenerative Systems (SSRS)*, pages 11–22, October 2003.

[24] V. Kiriansky, D. Bruening, and S. Amarasinghe. Secure Execution Via Program Shepherding. In *Proceedings of the 11th USENIX Security Symposium*, August 2002.

[25] Calvin Ko, Timothy Fraser, Lee Badger, and Douglas Kilpatrick. Detecting and Countering System Intrusions Using Software Wrappers. In *Proceedings of the 9th USENIX Security Symposium*, 2000.

[26] O. Patrick Kreidl and Tiffany M. Frazier. Feedback Control Applied to Survivability: A Host-Based Autonomic Defense System. *IEEE Transactions on Reliability*, 2002.

[27] Leonard J. LaPadula. Intrusion Reaction: Recommendations for Obtaining Reaction Capabilities. Technical report, September 1998.

[28] Prasad Naldurg and Roy H. Campbell. Dynamic Access Control: Preserving Safety and Trust for Network Defense Operations. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT) 2003*, June 2003.

[29] D. Nojiri, J. Rowe, and K. Levitt. Cooperative Response Strategies for Large Scale Attack Mitigation. In *Proceedings of The Third DARPA Information Survivability Conference and Exposition (DISCEX III),*, April 2003.

[30] Adam J. O'Donnell and Harish Sethu. On Achieving Software Diversity for Improved Network Security Using Distributed Coloring Algoritms. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS)*, pages 121–131, October 2004.

[31] Richard E. Overill. How Re(Pro)active Should an IDS Be? In *Proceedings of the 1st International Workshop on Recent Advances in Intrusion Detection (RAID)*, September 1998.

[32] Partha Pal, Franklin Webber, and Richard Schantz. Survival by Defense-Enabling. In *Proceedings of the New Security Paradigms Workshop (NSPW)*, pages 71–78, September 2001.

[33] Partha Pal, Franklin Webber, Richard Schantz, and Joseph P. Loyall. Intrusion Tolerant Systems. In *Proceedings of the ISW*, 2000.

[34] Phillip Porras, L. Briesemeister, K. Skinner, K. Levitt, J. Rowe, and Y. A. Ting. A Hybrid Quarantine Defense. In *Proceedings of the ACM CCS Workshop on Rapid Malcode (WORM) 2004*, October 2004.

[35] M. Prasad and T. Chiueh. A Binary Rewriting Defense Against Stack-based Buffer Overflow Attacks. In *Proceedings of the USENIX Annual Technical Conference*, June 2003.

[36] Niels Provos. Improving Host Security with System Call Policies. In *Proceedings of the 12th USENIX Security Symposium*, pages 207–225, August 2003.

[37] Marcelo Reis, Fabricio Paula, Diego Fernandes, and Paulo Geus. A Hybrid IDS Architecture Based on the Immune System. In *XXX*, 2002.

[38] James C. Reynolds and Lawrence A. Clough. Continual Repair for Windows Using the Event Log. In *Proceedings of the 1st ACM Workshop on Survivable and Self-Regenerative Systems (SSRS)*, pages 99–104, October 2003.

[39] James C. Reynolds, James Just, Larry Clough, and Ryan Maglich. On-Line Intrusion Detection and Attack Prevention Using Diversity, Genrate-and-Test, and Generalization. In *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS) 2003*, 2003.

[40] M. Rinard, C. Cadar, D. Dumitran, D. Roy, T. Leu, and Jr. W Beebee. Enhancing server availability and security through failure-oblivious computing. In *Proceedings 6th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2004.

[41] Jeff Rowe, D. Schnackenberg, D. Darby, K. Levitt, C. Wee, D. Klotz, and J. Schatz. Intrusion Detection and Isolation Protocol: Automated Response to Attacks. In *Proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection (RAID)*, 1999.

[42] Riccardo Scandariato and John C. Knight. An Automated Defense System to Counter Internet Worms. In *DSN*, 2004.

[43] R. Sekar and P. Uppuluri. Synthesizing Fast Intrusion Prevention/Detection Systems from High-Level Specifications. In *Proceedings of the 8th USENIX Security Symposium*, 1999.

[44] H. Shacham, M. Page, B. Pfaff, E.J. Goh, N. Modadugu, and D. Boneh. On the Effectiveness of Address-Space Randomization. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS)*, pages 298–307, October 2004.

[45] S. Sidiroglou and A. D. Keromytis. A Network Worm Vaccine Architecture. In *Proceedings of the IEEE Workshop on Enterprise Technologies: Infrastructure for Collaborative Enterprises (WETICE), Workshop on Enterprise Security*, pages 220–225, June 2003.

[46] A. Somayaji and S. Forrest. Automated Response Using System-Call Delays. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[47] A. Somayaji, S. Hofmeyer, and S. Forrest. Principles of a Computer Immune System. In *Proceedings of the New Security Paradigms Workshop (NPSW)*, pages 75–82, 1998.

[48] Anil B. Somayaji. Operating System Stability and Security through Process Homeostasis, July 2002.

[49] John D. Strunk, Garth R. Goodson, Adam G. Pennington, Craig Soules, and Gregory Ganger. Intrusion Detection, Diagnosis, and Recovery with Self-Securing Storage. Technical report, May 2002.

[50] Helen J. Wang, Chuanxiong Guo, Daniel R. Simon, and Alf Zugenmaier. Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits. In *Proceedings of the ACM SIGCOMM*, August 2004.

[51] Ke Wang and Salvatore J. Stolfo. Anomalous Payload-based Network Intrusion Detection. In *Proceedings of the Recent Advances in Intrusion Detection (RAID) Conference*, September 2004.

[52] Welch. Strike Back: Offensive Actions in Information Warfare. In *Proceedings of the New Security Paradigms Workshop (NSPW)*, 1999.

[53] A. Wolf, D. Heimbigner, A. Carzaniga, J. Knight, P. Devenbu, and M. Gertz. Bend, Don't Break: Using Reconfiguration to Achieve Survivability. In *Proceedings of the 3rd Information Survivability Workshop*, pages 187–190, October 2000.

[54] X.Wang, D. Reeves, and S.F. Wu. Tracing Based Active Intrusion Response. *Journal of Information Warfare*, 1:50–61, September 2001.